# Paralelización con OpenMP

Computación de Altas Prestaciones

Máster Universitario en Ingeniería Informática

Emilio Cobos Álvarez
Miguel Barranquero Díez

## Índice



## 1. Introducción

Se ha paralelizado el ejemplo de `waterwave` siguiendo el modelo propuesto anteriormente, usando OpenMP. Se ha paralelizado tanto la versión original, como la versión con MPI y nosequé.

## 2. Importancia de la paralelización híbrida MPI+OpenMP

La combinación de MPI (Message Passing Interface) y OpenMP representa un enfoque de paralelización multinivel que permite explotar eficientemente la arquitectura jerárquica de los sistemas de computación modernos. Este modelo híbrido es particularmente relevante en clústeres de computación de altas prestaciones, donde cada nodo contiene múltiples procesadores con memoria compartida, pero los nodos entre sí tienen memoria distribuida.

MPI proporciona paralelismo a nivel de proceso mediante paso de mensajes, siendo ideal para sistemas con memoria distribuida donde cada proceso tiene su propio espacio de direcciones. Por otro lado, OpenMP ofrece paralelismo a nivel de hilo dentro de un mismo proceso, aprovechando la memoria compartida dentro de un nodo. La estrategia híbrida permite usar MPI para comunicación entre nodos y OpenMP para paralelización dentro de cada nodo, reduciendo el número de procesos MPI necesarios y, por tanto, el overhead de comunicación entre procesos.

Esta aproximación presenta varias ventajas significativas. En primer lugar, reduce la sobrecarga de comunicación: al disminuir el número de procesos MPI y usar hilos OpenMP dentro de cada nodo, se minimiza el tráfico de red entre nodos. En segundo lugar, mejora la utilización de memoria: los hilos OpenMP comparten el espacio de direcciones del proceso, evitando la duplicación de datos que ocurriría con múltiples procesos MPI en el mismo nodo. Finalmente, proporciona mayor flexibilidad en la configuración de recursos, permitiendo ajustar la propor-

ción de procesos MPI y hilos OpenMP según las características del problema y del hardware disponible.

No obstante, la paralelización híbrida también introduce complejidades adicionales. Es necesario considerar la compatibilidad de niveles de threading de MPI (como `MPI_THREAD_FUNNELED` o `MPI_THREAD_MULTIPLE`), gestionar adecuadamente el pinning de procesos y hilos a núcleos físicos para evitar contención, y equilibrar la carga de trabajo entre ambos niveles de paralelismo. Además, el debugging y análisis de rendimiento se vuelve más complejo al tener que considerar dos modelos de programación simultáneamente.

# 3. Cambios al código

Afortunadamente ambas versiones son trivialmente paralelizables con OpenMP con mínimos cambios, ya que originalmente paralelizamos el bucle for de manera similar.

## 3.1. Paralelización del código con `pthreads`

Todos los cambios necesarios fueron añadir un flag para usar OpenMP en vez de nuestro `ThreadPool`, y una entrada nueva al `Makefile` para compilar con `OpenMP`.

```
diff --git a/Makefile b/Makefile
index e802619..2a21d4e 100644
--- a/Makefile
+++ b/Makefile
@@ -13,6 +13,9 @@ default: waterwave_cpp waterwave_mpi
 waterwave_cpp: waterwave.cpp
  $(CXX) $< -o $@ $(CXXFLAGS)

+waterwave_omp: waterwave.cpp
+ $(CXX) $< -o $@ $(CXXFLAGS) -DOMP -fopenmp
+
 waterwave_mpi: waterwave_mpi.cpp
  mpic++ $< -o $@ $(CXXFLAGS)

diff --git a/waterwave.cpp b/waterwave.cpp
index 1e80ee9..41bf227 100644
--- a/waterwave.cpp
+++ b/waterwave.cpp
@@ -12,6 +12,10 @@

 using std::size_t;

+#ifdef OMP
+#include <omp.h>
+#endif
+
 template <typename T> struct Range {
   T m_start;
   T m_end;
@@ -70,6 +74,8 @@ inline size_t index_for(size_t x, size_t y, size_t
size) {
     }
\
```

```
   } while (false)

+#ifndef OMP
+
 struct ThreadPool {
   struct ThreadState {
     size_t m_index;
@@ -176,6 +182,7 @@ struct ThreadPool {
 };

 ThreadPool ThreadPool::s_instance;
+#endif

 struct Grid {
   size_t size{};
@@ -299,6 +306,14 @@ struct SimulationState {
   template <typename F>
   void maybe_parallelize_loop(Range<size_t> x_range, Range<size_t>
y_range,
                               F callback) {
+#ifdef OMP
+    #pragma omp parallel for
+    for (size_t i = x_range.m_start; i < x_range.m_end; ++i) {
+      for (size_t j : y_range) {
+        callback(i, j);
+      }
+    }
+#else
     auto increment = x_range.size() / THREADS;
     for (size_t i = 0; i < THREADS; ++i) {
       size_t start = x_range.m_start + (i * increment);
@@ -315,6 +330,7 @@ struct SimulationState {
       });
     }
     ThreadPool::join();
+#endif
   }

   void reflect_boundaries() {
@@ -461,6 +477,9 @@ struct SimulationState {
 };

 int main() {
+#ifdef OMP
+  omp_set_num_threads(THREADS);
+#endif
   SimulationState state;
   while (state.tick()) {
     // keep going...
```

## 3.2. Paralelización del código MPI

Similarmente, los cambios necesario para la paralización híbrida son mínimos:

```
diff --git a/Makefile b/Makefile
index 2a21d4e..14ba73a 100644
--- a/Makefile
+++ b/Makefile
@@ -19,6 +19,9 @@ waterwave_omp: waterwave.cpp
 waterwave_mpi: waterwave_mpi.cpp
  mpic++ $< -o $@ $(CXXFLAGS)

+waterwave_mpi_omp: waterwave_mpi.cpp
+ mpic++ $< -o $@ $(CXXFLAGS) -DOMP -fopenmp
+
 mpismoketest: waterwave_mpi
  DROPS=$(DROPS) mpirun waterwave_mpi

diff --git a/waterwave_mpi.cpp b/waterwave_mpi.cpp
index e5b9fc4..79e1b47 100644
--- a/waterwave_mpi.cpp
+++ b/waterwave_mpi.cpp
@@ -11,6 +11,9 @@
 #include <span>
 #include <type_traits>
 #include <vector>
+#ifdef OMP
+#include <omp.h>
+#endif

 struct Point {
   uint32_t x = 0;
@@ -106,6 +109,7 @@ static uint32_t get_size_from_env(const char
*name, uint32_t def) {
   static const uint32_t name = get_size_from_env(#name, def)

 ENV_SIZE(GRID_SIZE, 64);
+ENV_SIZE(OMP_THREADS, 1);
 ENV_SIZE(MAX_DROPS, 5);     // Maximum number of drops
 ENV_SIZE(DROP_STEP, 500);   // Steps between drops
 ENV_SIZE(STEP_COUNT, 1000); // Total steps
@@ -227,7 +231,10 @@ struct SimulationState {
   template <typename F>
   void maybe_parallelize_loop(Range<uint32_t> x_range,
Range<uint32_t> y_range,
                               F callback) {
-    for (uint32_t i : x_range) {
+#ifdef OMP
+#pragma omp parallel for
+#endif
+    for (uint32_t i = x_range.m_start; i < x_range.m_end; ++i) {
       for (uint32_t j : y_range) {
         callback(i, j);
       }
@@ -501,7 +508,18 @@ struct SimulationState {
 };
```

```
 int main(int argc, char **argv) {
+  int provided = 0;
+#ifdef OMP
+  MPI_INFALLIBLE(MPI_Init_thread(&argc, &argv, MPI_THREAD_FUNNELED,
&provided));
+  if (provided < MPI_THREAD_FUNNELED) {
+    assert(false && "Couldn't do multi-threaded MPI?");
+    abort();
+  }
+  omp_set_num_threads(OMP_THREADS);
+#else
   MPI_INFALLIBLE(MPI_Init(&argc, &argv));
+#endif
+
   MPI_INFALLIBLE(MPI_Comm_size(MPI_COMM_WORLD,
&MPI::g_proc_count));
   MPI_INFALLIBLE(MPI_Comm_rank(MPI_COMM_WORLD, &MPI::g_pid));
```

## 4. Comparaciones

Se ha modificado el script de las entregas anteriores para comparar las versiones normales y con OpenMP para también comparar OpenMP y OpenMP+MPI. Para ello se ha creado un script llamado `launch_mpi_omp` que se encarga de traducir la variable `Threads` en un mix de nodos MPI y hilos OpenMP:

```
#!/usr/bin/env bash

set -euxo pipefail
export MPI_PROCS=$((THREADS / 2))
if [ $MPI_PROCS -eq 0 ]; then
  # Deal with THREADS=1
  export MPI_PROCS=1
fi
export OMP_THREADS=$((THREADS / MPI_PROCS))
mpirun \
  -n $MPI_PROCS \
  -x STEP_COUNT \
  -x DUMP_EVERY \
  -x GRID_SIZE \
  ./waterwave_mpi_omp
```

Tras ejecutar la misma comparación, se obtiene el siguiente rendimiento para la matriz de 512:

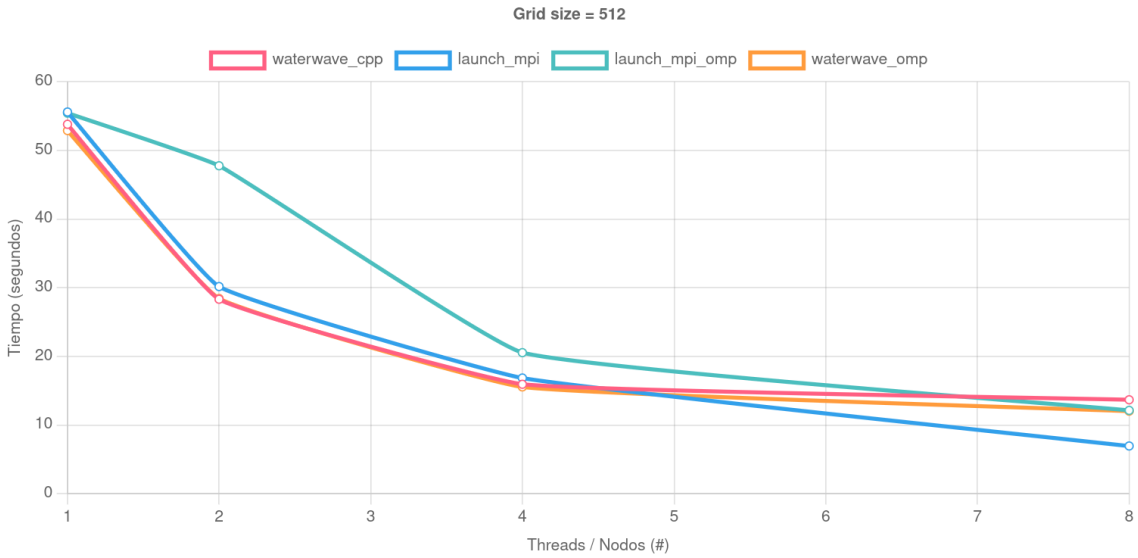


Figura 1: Datos con matriz de 512

Se observa una anomalía con la versión híbrida, que es más lenta de lo esperado. Esta anomalía es reproducible, y el perfil de rendimiento[1] no muestra nada anómalo, parece que el trabajo se está distribuyendo como es de esperar.

Investigamos con `perf stat`, comparada con la versión de `OpenMP` normal:

```
$ THREADS=2 GRID_SIZE=512 /usr/bin/time -o t.time perf stat ./
launch_mpi_omp >t.out
Performance counter stats for './launch_mpi_omp':

       95121965923      task-clock                       #    1.949
CPUs utilized
             19211      context-switches                 #
201.962 /sec
                56      cpu-migrations                   #
0.589 /sec
              8511      page-faults                      #
89.475 /sec
      644749254139      instructions                     #    1.41
insn per cycle
                                                  #    0.01  stalled
cycles per insn
      458205714784      cycles                           #    4.817
GHz
        3323560348      stalled-cycles-frontend          #    0.73%
frontend cycles idle
        8621620251      branches                         #   90.638
M/sec
          20223156      branch-misses                    #    0.23%
of all branches

      48.797620834 seconds time elapsed
```

[1] https://share.firefox.dev/3LetGdr

```
      94.612146000 seconds user
       0.208231000 seconds sys
```

```
$ THREADS=2 GRID_SIZE=512 /usr/bin/time -o t.time perf stat ./
waterwave_omp >t.out
 Performance counter stats for './waterwave_omp':

       56906275617      task-clock                       #    1.991
CPUs utilized
               405      context-switches                 #
7.117 /sec
               133      cpu-migrations                   #
2.337 /sec
              2487      page-faults                      #
43.703 /sec
      529001639388      instructions                     #    2.01
insn per cycle
                                                  #    0.00  stalled
cycles per insn
      263829534379      cycles                           #    4.636
GHz
        1105091784      stalled-cycles-frontend          #    0.42%
frontend cycles idle
        9042363379      branches                         #  158.899
M/sec
          21431108      branch-misses                    #    0.24%
of all branches

      28.584208088 seconds time elapsed

      56.723608000 seconds user
       0.013957000 seconds sys
```

La mayor anomalía se encuentra en `context-switches`, que son mucho mayores en la versión híbrida. Esto hace sospechar inmediatamente de un conflicto de prioridades.

OpenMPI fija el proceso completo a un core, y este procesador no tiene hyper-threading, por lo que ambos hilos compiten por la misma CPU.

Se investigó, y se confirmó que pasando `--bind-to none` a `mpirun` el rendimiento vuelve a los niveles esperados:

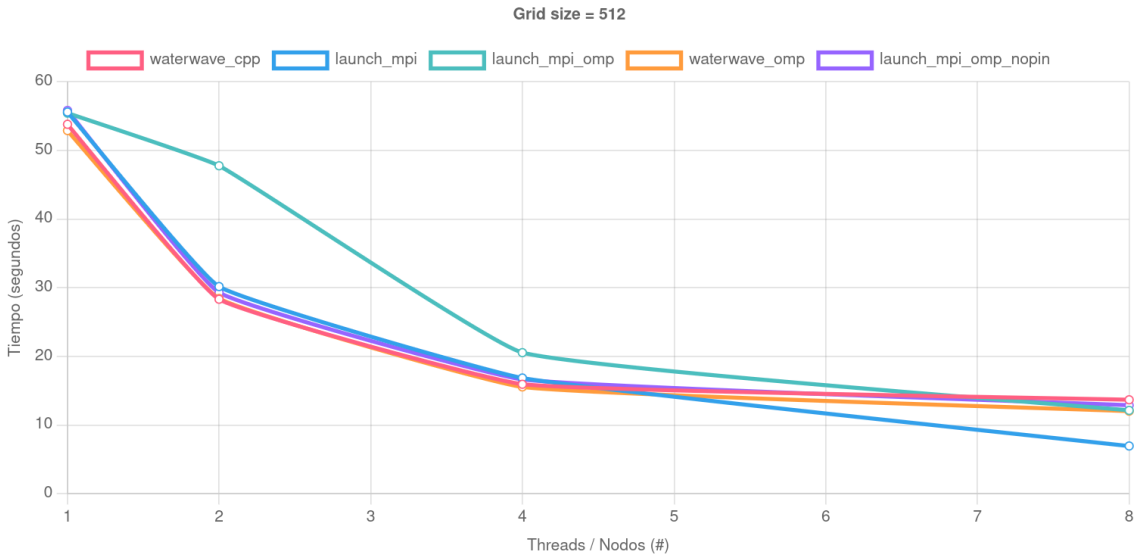


Figura 2: Datos con matriz de 512

## 4.1. Datos completos (tras arreglar el pinning)

| Programa | Tamaño del grid | Hilos | Tiempo |
|---|---|---|---|
| launch_mpi | 128 | 1 | 0:02.98 |
| launch_mpi | 128 | 2 | 0:02.25 |
| launch_mpi | 128 | 4 | 0:01.92 |
| launch_mpi | 128 | 8 | 0:01.89 |
| launch_mpi | 16 | 1 | 0:01.48 |
| launch_mpi | 16 | 2 | 0:01.57 |
| launch_mpi | 16 | 4 | 0:01.51 |
| launch_mpi | 16 | 8 | 0:01.53 |
| launch_mpi | 256 | 1 | 0:08.44 |
| launch_mpi | 256 | 2 | 0:04.87 |
| launch_mpi | 256 | 4 | 0:03.24 |
| launch_mpi | 256 | 8 | 0:02.74 |
| launch_mpi | 32 | 1 | 0:01.58 |
| launch_mpi | 32 | 2 | 0:01.54 |
| launch_mpi | 32 | 4 | 0:01.52 |
| launch_mpi | 32 | 8 | 0:01.56 |
| launch_mpi | 512 | 1 | 0:55.36 |
| launch_mpi | 512 | 2 | 0:28.21 |
| launch_mpi | 512 | 4 | 0:16.44 |
| launch_mpi | 512 | 8 | 0:06.72 |
| launch_mpi | 64 | 1 | 0:01.86 |
| launch_mpi | 64 | 2 | 0:01.68 |

| Programa | Tamaño del grid | Hilos | Tiempo |
|---|---|---|---|
| launch_mpi | 64 | 4 | 0:01.61 |
| launch_mpi | 64 | 8 | 0:01.66 |
| launch_mpi | 8 | 1 | 0:01.58 |
| launch_mpi | 8 | 2 | 0:01.48 |
| launch_mpi | 8 | 4 | 0:01.49 |
| launch_mpi | 8 | 8 | 0:01.55 |
| launch_mpi_omp | 128 | 1 | 0:03.06 |
| launch_mpi_omp | 128 | 2 | 0:02.35 |
| launch_mpi_omp | 128 | 4 | 0:01.95 |
| launch_mpi_omp | 128 | 8 | 0:02.00 |
| launch_mpi_omp | 16 | 1 | 0:01.50 |
| launch_mpi_omp | 16 | 2 | 0:01.59 |
| launch_mpi_omp | 16 | 4 | 0:01.57 |
| launch_mpi_omp | 16 | 8 | 0:01.60 |
| launch_mpi_omp | 256 | 1 | 0:08.53 |
| launch_mpi_omp | 256 | 2 | 0:05.44 |
| launch_mpi_omp | 256 | 4 | 0:03.45 |
| launch_mpi_omp | 256 | 8 | 0:03.09 |
| launch_mpi_omp | 32 | 1 | 0:01.56 |
| launch_mpi_omp | 32 | 2 | 0:01.67 |
| launch_mpi_omp | 32 | 4 | 0:01.61 |
| launch_mpi_omp | 32 | 8 | 0:01.69 |
| launch_mpi_omp | 512 | 1 | 0:55.82 |
| launch_mpi_omp | 512 | 2 | 0:29.29 |
| launch_mpi_omp | 512 | 4 | 0:16.54 |
| launch_mpi_omp | 512 | 8 | 0:12.82 |
| launch_mpi_omp | 64 | 1 | 0:01.85 |
| launch_mpi_omp | 64 | 2 | 0:01.77 |
| launch_mpi_omp | 64 | 4 | 0:01.64 |
| launch_mpi_omp | 64 | 8 | 0:01.70 |
| launch_mpi_omp | 8 | 1 | 0:01.47 |
| launch_mpi_omp | 8 | 2 | 0:01.52 |
| launch_mpi_omp | 8 | 4 | 0:01.51 |
| launch_mpi_omp | 8 | 8 | 0:01.53 |
| waterwave_cpp | 128 | 1 | 0:02.13 |
| waterwave_cpp | 128 | 2 | 0:01.35 |
| waterwave_cpp | 128 | 4 | 0:00.97 |

| Programa | Tamaño del grid | Hilos | Tiempo |
|---|---|---|---|
| waterwave_cpp | 128 | 8 | 0:01.56 |
| waterwave_cpp | 16 | 1 | 0:00.42 |
| waterwave_cpp | 16 | 2 | 0:00.42 |
| waterwave_cpp | 16 | 4 | 0:00.46 |
| waterwave_cpp | 16 | 8 | 0:00.68 |
| waterwave_cpp | 256 | 1 | 0:07.73 |
| waterwave_cpp | 256 | 2 | 0:04.43 |
| waterwave_cpp | 256 | 4 | 0:02.72 |
| waterwave_cpp | 256 | 8 | 0:03.09 |
| waterwave_cpp | 32 | 1 | 0:00.52 |
| waterwave_cpp | 32 | 2 | 0:00.59 |
| waterwave_cpp | 32 | 4 | 0:00.60 |
| waterwave_cpp | 32 | 8 | 0:01.00 |
| waterwave_cpp | 512 | 1 | 0:53.70 |
| waterwave_cpp | 512 | 2 | 0:27.82 |
| waterwave_cpp | 512 | 4 | 0:15.51 |
| waterwave_cpp | 512 | 8 | 0:13.29 |
| waterwave_cpp | 64 | 1 | 0:00.82 |
| waterwave_cpp | 64 | 2 | 0:00.62 |
| waterwave_cpp | 64 | 4 | 0:00.66 |
| waterwave_cpp | 64 | 8 | 0:01.30 |
| waterwave_cpp | 8 | 1 | 0:00.46 |
| waterwave_cpp | 8 | 2 | 0:00.39 |
| waterwave_cpp | 8 | 4 | 0:00.41 |
| waterwave_cpp | 8 | 8 | 0:00.59 |
| waterwave_omp | 128 | 1 | 0:01.75 |
| waterwave_omp | 128 | 2 | 0:01.08 |
| waterwave_omp | 128 | 4 | 0:00.75 |
| waterwave_omp | 128 | 8 | 0:01.22 |
| waterwave_omp | 16 | 1 | 0:00.27 |
| waterwave_omp | 16 | 2 | 0:00.37 |
| waterwave_omp | 16 | 4 | 0:00.42 |
| waterwave_omp | 16 | 8 | 0:00.54 |
| waterwave_omp | 256 | 1 | 0:07.10 |
| waterwave_omp | 256 | 2 | 0:07.64 |
| waterwave_omp | 256 | 4 | 0:02.35 |
| waterwave_omp | 256 | 8 | 0:02.39 |

| Programa | Tamaño del grid | Hilos | Tiempo |
|---|---|---|---|
| waterwave_omp | 32 | 1 | 0:00.33 |
| waterwave_omp | 32 | 2 | 0:00.37 |
| waterwave_omp | 32 | 4 | 0:00.54 |
| waterwave_omp | 32 | 8 | 0:00.89 |
| waterwave_omp | 512 | 1 | 0:53.45 |
| waterwave_omp | 512 | 2 | 0:28.15 |
| waterwave_omp | 512 | 4 | 0:15.26 |
| waterwave_omp | 512 | 8 | 0:11.72 |
| waterwave_omp | 64 | 1 | 0:00.64 |
| waterwave_omp | 64 | 2 | 0:00.46 |
| waterwave_omp | 64 | 4 | 0:00.42 |
| waterwave_omp | 64 | 8 | 0:01.19 |
| waterwave_omp | 8 | 1 | 0:00.25 |
| waterwave_omp | 8 | 2 | 0:00.27 |
| waterwave_omp | 8 | 4 | 0:00.28 |
| waterwave_omp | 8 | 8 | 0:00.33 |

# 5. Otras observaciones y conclusiones

Se ha observado que OpenMP funciona mejor que nuestra versión manual cuando no se usa el procesador en modo «performance» (para realizar las medidas se ha usado el `performance` governor[2]). Esto se atribuye a la combinación del uso de *spin-locks* (que evita que el clock del procesador se regule) y *pinning* de hilos a núcleos particulares que GOMP hace por defecto (ver OMP_PROC_BIND[3]). Nótese que `OMP_PLACES` existe por defecto, lo cual se ha verificado usando `OMP_DISPLAY_ENV=VERBOSE`[4].

También se ha observado que la solución con MPI funciona mejor en este hardware. Esto es porque para el caso de 8 nodos, MPI consigue pinear cada uno a un núcleo de alto rendimiento de este procesador, lo que combinado con el algoritmo (que mantiene sólo la fracción de la matriz requerida en memoria) maximiza la utilidad de las cachés.

Aún así, la versión híbrida sería muy útil para ordenadores con memoria no compartida, en vez de requerir múltiples nodos de MPI en el mismo ordenador físico.

[2] https://linux.die.net/man/1/cpupower-frequency-set
[3] https://gcc.gnu.org/onlinedocs/libgomp/OMP_005fPROC_005fBIND.html
[4] https://www.openmp.org/spec-html/5.0/openmpse60.html